APPENDIX A

```
//COPYRIGHT 2000 CISCO SYSTEMS, INC.,

//TCC BLSR STATE MACHINE PSEUDO-CODE

States:
    UNPROVISIONED_STATE,
    TOPOLOGY_RDY_STATE,
    PROVISIONING_RDY_STATE,
    STANDBY_STATE,
    IDLE_STATE,
    WAIT_RM_ACK_STATE,
    WAIT_BLSR_TBL_RSP_STATE,
    WAIT_NODE_ID_ACK_STATE,
    WAIT_ENABLE_SWITCH_RSP_STATE

Events:
    TOPOLOGY_CHANGE,
    PROVISIONING,
    LOCAL_XC_CHANGE,
    LOCAL_NODE_ID_CHANGE,
    LOCAL_RING_ID_CHANGE,
    REMOTE_XC_CHANGE,
    REMOTE_NODE_ID_CHANGE,
    REMOTE_RM_CHANGE,
    RDY_TO_SWITCH_REQ,
    BLSR_TBL_REQ,
    BLSR_TBL_RSP,
    NODE_ID_ACK,
    RM_ACK,
    RDY_TO_SWITCH_ACK,
    ENABLE_SWITCH,
    XC_ENABLE_SWITCH_REQ,
    USER_ACCEPT_RM,

Initial state UNPROVISIONED_STATE;

////////////////////////////////////////////////////////
// UNPROVISIONED_STATE
////////////////////////////////////////////////////////
for(UNPROVISIONED_STATE;
    TOPOLOGY_CHANGE)
{
  //topo change given by OSPF. Simply go to next state
  NEXT_STATE(TOPOLOGY_RDY_STATE);
}

for(UNPROVISIONED_STATE;
    PROVISIONING)
{
  initialize local Payload table;
  initialize squelch table;
  initialize local add/drop tables;

  if this TCC is not active {
    blsrActive = FALSE;
    readyToSwitch = FALSE;
    NEXT_STATE(STANDBY_STATE);
  }
```

```
      if number of nodes in ring > 1 {
        delete all communication session for ring id ringId;
        send a request for other nodes' blsr tables;
        NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
      }

      NEXT_STATE(PROVISIONING_RDY_STATE);

}

for(UNPROVISIONED_STATE;
    REMOTE_XC_CHANGE, REMOTE_NODE_ID_CHANGE, REMOTE_RM_CHANGE,
    RDY_TO_SWITCH_REQ, XC_ENABLE_SWITCH_REQ, BLSR_TBL_REQ)
{
  //Ignore
  SAME_STATE;
}

for(UNPROVISIONED_STATE;
    LOCAL_XC_CHANGE, LOCAL_NODE_ID_CHANGE, LOCAL_RING_ID_CHANGE,
    BLSR_TBL_RSP, NODE_ID_ACK, RM_ACK, RDY_TO_SWITCH_ACK,
    ENABLE_SWITCH, USER_ACCEPT_RM)
{
  STATE_ERROR;
}

////////////////////////////////////////////////////////
// TOPOLOGY_RDY_STATE
////////////////////////////////////////////////////////
for(TOPOLOGY_RDY_STATE;
    TOPOLOGY_CHANGE)
{
  //topo change given by OSPF. Simply go to next state
  SAME_STATE;
}

for(TOPOLOGY_RDY_STATE;
    PROVISIONING)
{
  initialize local payload table;
  initialize squelch table;
  initialize local add/drop tables;

  if TCC is not active {
    blsrActive = FALSE;
    readyToSwitch = FALSE;
    NEXT_STATE(STANDBY_STATE);
  }

  // To force update of all tables once topo change is performed
  // set localXcChangeQ to TRUE;
  // We cannot guarantee that topo update will result in new
  // Ring map, so we need to force table generation.
  localXcChangeQ = TRUE;

  process OSPF changes;
  if OSPF changes lead to new ring map {
    NEXT_STATE(IDLE_STATE);
  }
```

```
    NEXT_STATE(PROVISIONING_RDY_STATE);
  }

  for(TOPOLOGY_RDY_STATE;
      REMOTE_XC_CHANGE, REMOTE_NODE_ID_CHANGE, REMOTE_RM_CHANGE,
      RDY_TO_SWITCH_REQ, XC_ENABLE_SWITCH_REQ,  BLSR_TBL_REQ)
  {
    //Ignore
    SAME_STATE;
  }

  for(TOPOLOGY_RDY_STATE;
      LOCAL_XC_CHANGE, LOCAL_NODE_ID_CHANGE, LOCAL_RING_ID_CHANGE,
      BLSR_TBL_RSP,  NODE_ID_ACK, RM_ACK, RDY_TO_SWITCH_ACK,
      ENABLE_SWITCH, USER_ACCEPT_RM)
  {
    STATE_ERROR;
  }

  ////////////////////////////////////////////////////
  // PROVISIONING_RDY_STATE
  ////////////////////////////////////////////////////
  for(PROVISIONING_RDY_STATE;
      TOPOLOGY_CHANGE)
  {
    process OPSF change;
    if OSPF changes lead to new ring map {
       NEXT_STATE(IDLE_STATE);
    }
    SAME_STATE;
  }

  for(PROVISIONING_RDY_STATE;
      PROVISIONING)
  {
    if TCC is not active
    {
      blsrActive = FALSE;
      readyToSwitch = FALSE;
      terminate all communication sessions with other nodes;
      NEXT_STATE(STANDBY_STATE);
    }

    SAME_STATE;
  }

  for(PROVISIONING_RDY_STATE;
      LOCAL_XC_CHANGE)
  {
    initialize local payload table;
    initialize squelch table;
    initialize add/drop tables;

    SAME_STATE;
  }

  for(PROVISIONING_RDY_STATE;
      LOCAL_NODE_ID_CHANGE)
  {
    SAME_STATE;
  }
```

```
for(PROVISIONING_RDY_STATE;
    LOCAL_RING_ID_CHANGE)
{
  SAME_STATE;
}

for(PROVISIONING_RDY_STATE;
    REMOTE_RM_CHANGE)
{
  process ring map update from other node;
  send acknowledgment to node;
  terminate all communication sessions with other nodes;
  Send a blsr table request to all nodes;
  NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
}

for(PROVISIONING_RDY_STATE;
    BLSR_TBL_REQ)
{
  process the blsr table request;
  send a blsr table response;
  SAME_STATE;
}

for(PROVISIONING_RDY_STATE;
    REMOTE_XC_CHANGE, REMOTE_NODE_ID_CHANGE, RDY_TO_SWITCH_REQ,
    XC_ENABLE_SWITCH_REQ)
{
  // Ignore
  SAME_STATE;
}

for(PROVISIONING_RDY_STATE;
    BLSR_TBL_RSP,  NODE_ID_ACK,
    RM_ACK, RDY_TO_SWITCH_ACK, ENABLE_SWITCH,
    USER_ACCEPT_RM)
{
  STATE_ERROR;
}

////////////////////////////////////////////////////
// STANDBY_STATE
////////////////////////////////////////////////////
for(STANDBY_STATE;
    TOPOLOGY_CHANGE)
{
  //topo change stored before event. Simply go to next state
  SAME_STATE;
}

for(STANDBY_STATE;
    PROVISIONING)
{
  //misc.
  if TCC is active {
    blsrActive = TRUE;

    initialize the local payload table;
    initialize the squelch table;
    initialize the add/drop tables;
```

```
        if number of nodes in ring > 1 {
          terminate all communication sessions;
          send a request for blsr tables to all nodes in ring;
          NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
        }

        NEXT_STATE(PROVISIONING_RDY_STATE);
      }
      else {
        SAME_STATE;
      }
    }

    for(STANDBY_STATE;
        XC_ENABLE_SWITCH_REQ)
    {
      //Ignore
      SAME_STATE;
    }

    for(STANDBY_STATE;
        LOCAL_XC_CHANGE, LOCAL_NODE_ID_CHANGE, LOCAL_RING_ID_CHANGE,
        REMOTE_XC_CHANGE, REMOTE_NODE_ID_CHANGE, REMOTE_RM_CHANGE,
        RDY_TO_SWITCH_REQ, BLSR_TBL_REQ, BLSR_TBL_RSP,
         NODE_ID_ACK, RM_ACK, RDY_TO_SWITCH_ACK,
        ENABLE_SWITCH, USER_ACCEPT_RM)
    {
      STATE_ERROR;
    }

    ////////////////////////////////////////////////////
    // IDLE_STATE,
    ////////////////////////////////////////////////////
    for(IDLE_STATE;
        TOPOLOGY_CHANGE)
    {
      process OSPF changes;
      if OSPF changes lead to new ring map {
        NEXT_STATE(IDLE_STATE);
      }
      SAME_STATE;
    }

    for(IDLE_STATE;
        PROVISIONING)
    {
      if TCC is not active
      {
        blsrActive = FALSE;
        readyToSwitch = FALSE;
        terminate all communication sessions;
        NEXT_STATE(STANDBY_STATE);
      }

      SAME_STATE;
    }

    for(IDLE_STATE;
        LOCAL_XC_CHANGE)
    {
```

```
      initialize local payload table;
      initialize squelch table;
      initialize add/drop tables;

 5    if number of nodes in ring > 1 {
        send to all nodes in ring that local cross-connect table was
modified;
        send a request for blsr tables to all nodes in ring;
        NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
10    }

      SAME_STATE;
    }

15  for(IDLE_STATE;
        LOCAL_NODE_ID_CHANGE)
    {
      process new node id;
      if number of nodes in ring > 1 {
20      send to all nodes in ring the new node id;
        NEXT_STATE(WAIT_NODE_ID_ACK_STATE);
      }
      SAME_STATE;
    }
25
    for(IDLE_STATE;
        LOCAL_RING_ID_CHANGE)
    {
      terminate all communication sessions;
30    process latest OSPF changes;
      if new node id leads to a new ring map {
        NEXT_STATE(IDLE_STATE);
      }
      SAME_STATE;
35  }

    for(IDLE_STATE;
        REMOTE_XC_CHANGE)
    {
40    terminate all communication sessions;
      ask for blsr tables to all nodes in ring;
      NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
    }

45  for(IDLE_STATE;
        REMOTE_NODE_ID_CHANGE)
    {
      process new node id;
      send acknowledgement to node;
50    SAME_STATE;
    }

    for(IDLE_STATE;
        REMOTE_RM_CHANGE)
55  {
      process ring map change;
      send acknowledgment to node;
      terminate all communication sessions;
      send request for blsr tables to all nodes;
60    NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
    }
```

```
for(IDLE_STATE;
    RDY_TO_SWITCH_REQ)
{
  if readyToSwitch {
    send message to querying node that node is ready to switch.
  }
  SAME_STATE;
}

for(IDLE_STATE;
    BLSR_TBL_REQ)
{
  process request for blsr tables;
  send blsr tables to requesting node;
  SAME_STATE;
}

for(IDLE_STATE;
    ENABLE_SWITCH)
{
  send enable switch request to all nodes
  NEXT_STATE(WAIT_ENABLE_SWITCH_RSP_STATE);
}

for(IDLE_STATE;
    XC_ENABLE_SWITCH_REQ)
{
  if(!enableSwitchSentToXc && readyToSwitch) {
    send message to XCON that BLSR is ready to switch;
  }
  SAME_STATE;
}

for(IDLE_STATE;
    USER_ACCEPT_RM)
{
  if ring Map accepted by user is different than previous ring map
  {
    send accepted ring map to all nodes in ring.
    NEXT_STATE(WAIT_RM_ACK_STATE);
  }
  SAME_STATE;
}

for(IDLE_STATE;
    BLSR_TBL_RSP,  NODE_ID_ACK,
    RM_ACK, RDY_TO_SWITCH_ACK)
{
  STATE_ERROR;
}

////////////////////////////////////////////////////////
// WAIT_RM_ACK_STATE
////////////////////////////////////////////////////////
for(WAIT_RM_ACK_STATE;
    TOPOLOGY_CHANGE)
{
  // Queue event to make sure that we resume after topo change.
  process OSPF changes;
  if OSPF changes lead to new ring map {
```

```
      acceptRmQ = TRUE;
      NEXT_STATE(IDLE_STATE);
    }
    SAME_STATE;
}

for(WAIT_RM_ACK_STATE;
    PROVISIONING)
{
  if TCC is not active
  {
    blsrActive = FALSE;
    readyToSwitch = FALSE;
    terminate all communication sessions;
    NEXT_STATE(STANDBY_STATE);
  }

  SAME_STATE;
}

for(WAIT_RM_ACK_STATE;
    LOCAL_XC_CHANGE)
{
  localXcChangeQ = TRUE;
  SAME_STATE;
}
for(WAIT_RM_ACK_STATE;
    LOCAL_NODE_ID_CHANGE)
{
  process change of node id;
  if number of nodes in ring > 1 {
    acceptRmQ = TRUE;
    send new node id to all nodes in ring;
    NEXT_STATE(WAIT_NODE_ID_ACK_STATE);
  }
  SAME_STATE;
}

for(WAIT_RM_ACK_STATE;
    LOCAL_RING_ID_CHANGE)
{
  terminate all communication sessions;
  process latest OSPF changes;
  if new node id leads new ring map {
    NEXT_STATE(IDLE_STATE);
  }
  SAME_STATE;
}

for(WAIT_RM_ACK_STATE;
    REMOTE_XC_CHANGE)
{
  remoteXcChangeQ = TRUE;
  SAME_STATE;
}

for(WAIT_RM_ACK_STATE;
    REMOTE_NODE_ID_CHANGE)
{
  process new node id;
  send acknowledgement to node;
```

```
    SAME_STATE;
  }

  for(WAIT_RM_ACK_STATE;
      REMOTE_RM_CHANGE)
  {
    process ring map update;
    send acknowledgement to node;
    delete all communication sessions;
    send blsr tables to all nodes;
    NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
  }

  for(WAIT_RM_ACK_STATE;
      BLSR_TBL_REQ)
  {
    process blsr table request;
    send blsr tables to node;
    SAME_STATE;
  }

  for(WAIT_RM_ACK_STATE;
      RM_ACK)
  {
    if processing of ring map acknowledgement is successful
      terminate all communication sessions;
      send blsr table request to all nodes;
      NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
    }
    SAME_STATE;
  }

  for(WAIT_RM_ACK_STATE;
      USER_ACCEPT_RM)
  {
    STATE_ERROR;
  }

  for(WAIT_RM_ACK_STATE;
      RDY_TO_SWITCH_REQ)
  {
    // Ignore
    SAME_STATE;
  }

  for(WAIT_RM_ACK_STATE;
      XC_ENABLE_SWITCH_REQ)
  {
    xcEnableSwQ = TRUE;
    SAME_STATE;
  }

  for(WAIT_RM_ACK_STATE;
      BLSR_TBL_RSP,  NODE_ID_ACK, RDY_TO_SWITCH_ACK,
      ENABLE_SWITCH)
  {
    STATE_ERROR;
  }

  ////////////////////////////////////////////////////
  // WAIT_BLSR_TBL_RSP_STATE
```

```
////////////////////////////////////////////////////
for(WAIT_BLSR_TBL_RSP_STATE;
    TOPOLOGY_CHANGE)
{
  // Queue event to make sure that we resume after RM update.
  // Queue worse case event.
  process OSPF change
  if OSPF changes lead to new ring map {
    localXcChangeQ = TRUE;
    NEXT_STATE(IDLE_STATE);
  }
  SAME_STATE;
}

for(WAIT_BLSR_TBL_RSP_STATE;
    PROVISIONING)
{
  if TCC is not active
  {
    blsrActive = FALSE;
    readyToSwitch = FALSE;
    terminate all communication sessions;
    NEXT_STATE(STANDBY_STATE);
  }

  SAME_STATE;
}

for(WAIT_BLSR_TBL_RSP_STATE;
    LOCAL_XC_CHANGE)
{
  initialize local payload table;
  initialize squelch table;
  initialize add/drop tables;

  if number of nodes in ring > 1 {
    send cross-connect table change notification to all nodes in
ring;
    send blsr table request to all nodes in ring;
  }

  SAME_STATE;
}

for(WAIT_BLSR_TBL_RSP_STATE;
    LOCAL_NODE_ID_CHANGE)
{
  process node id change;
  if number of nodes in ring > 1 {
    localXcChangeQ = TRUE;
    send new node id to all nodes in ring;
    NEXT_STATE(WAIT_NODE_ID_ACK_STATE);
  }
  SAME_STATE;
}

for(WAIT_BLSR_TBL_RSP_STATE;
    LOCAL_RING_ID_CHANGE)
{
  terminate all communication sessions;
  process latest OSPF changes;
```

```
      if new ring Id lead to new ring map {
        NEXT_STATE(IDLE_STATE);
      }
      SAME_STATE;
5   }

    for(WAIT_BLSR_TBL_RSP_STATE;
        REMOTE_XC_CHANGE)
    {
10    terminate all communication sessions;
      send blsr table request to all nodes in ring;
      SAME_STATE;
    }

15  for(WAIT_BLSR_TBL_RSP_STATE;
        REMOTE_NODE_ID_CHANGE)
    {
      process node id change;
      send acknowledgement to node;
20    SAME_STATE;
    }

    for(WAIT_BLSR_TBL_RSP_STATE;
        REMOTE_RM_CHANGE)
25  {
      process ring map update;
      send ring map acknowledgement to node;
      terminate all communication sessions;
      send blsr table request to all nodes in ring;
30    SAME_STATE;
    }

    for(WAIT_BLSR_TBL_RSP_STATE;
        BLSR_TBL_REQ)
35  {
      process blsr table request;
      send blsr table response to requesting node;
      SAME_STATE;
    }
40
    for(WAIT_BLSR_TBL_RSP_STATE;
        BLSR_TBL_RSP)
    {
      process event;
45    if all responses have been received {
        if(!enableSwitchSentToXc) {
          readyToSwitch = TRUE;
          query all nodes in ring to know if they are ready to switch;
          NEXT_STATE(WAIT_ENABLE_SWITCH_RSP_STATE);
50      }
        NEXT_STATE(IDLE_STATE);
      }
      else {
        SAME_STATE;
55    }
    }

    for(WAIT_BLSR_TBL_RSP_STATE;
        USER_ACCEPT_RM)
60  {
```

```
      if ring map accepted by user is different from previous ring map
    {
       send a ring map update request to all nodes;
       NEXT_STATE(WAIT_RM_ACK_STATE);
 5     }
      SAME_STATE;
    }

    for(WAIT_BLSR_TBL_RSP_STATE;
10      RDY_TO_SWITCH_REQ)
    {
      // Ignore
      SAME_STATE;
    }
15
    for(WAIT_BLSR_TBL_RSP_STATE;
        XC_ENABLE_SWITCH_REQ)
    {
      xcEnableSwQ = TRUE;
20    SAME_STATE;
    }

    for(WAIT_BLSR_TBL_RSP_STATE;
         NODE_ID_ACK, RM_ACK,
25      RDY_TO_SWITCH_ACK, ENABLE_SWITCH)
    {
      STATE_ERROR;
    }

30  ////////////////////////////////////////////////////
    // WAIT_NODE_ID_ACK_STATE
    ////////////////////////////////////////////////////
    for(WAIT_NODE_ID_ACK_STATE;
        TOPOLOGY_CHANGE)
35  {
      // Queue event to make sure that we resume after topo change.
      // Queue worse case event.
      nodeIdChangeQ = TRUE;
      process OSPF changes;
40  }

    for(WAIT_NODE_ID_ACK_STATE;
        PROVISIONING)
    {
45    if TCC is not active
      {
        blsrActive = FALSE;
        readyToSwitch = FALSE;
        terminate all communication sessions;
50      NEXT_STATE(STANDBY_STATE);
      }

      SAME_STATE;
    }
55
    for(WAIT_NODE_ID_ACK_STATE;
        LOCAL_XC_CHANGE)
    {
      localXcChangeQ = TRUE;
60    SAME_STATE;
    }
```

```
for(WAIT_NODE_ID_ACK_STATE;
    LOCAL_NODE_ID_CHANGE)
{
  process local node id change;
  if number of nodes in ring > 1 {
    send node id update to all nodes in ring;
    NEXT_STATE(WAIT_NODE_ID_ACK_STATE);
  }
  SAME_STATE;
}

for(WAIT_NODE_ID_ACK_STATE;
    LOCAL_RING_ID_CHANGE)
{
  terminate all communication sessions;
  process latest OSPF changes;
  if new ring Id leads to new ring map {
    NEXT_STATE(IDLE_STATE);
  }
  SAME_STATE;
}

for(WAIT_NODE_ID_ACK_STATE;
    REMOTE_XC_CHANGE)
{
  remoteXcChangeQ = TRUE;
  SAME_STATE;
}
for(WAIT_NODE_ID_ACK_STATE;
    REMOTE_NODE_ID_CHANGE)
{
  process node id update;
  send acknowledgement to node;
  SAME_STATE;
}

for(WAIT_NODE_ID_ACK_STATE;
    REMOTE_RM_CHANGE)
{
  process ring map update;
  send ring map acknowledgement;
  send blsr table request to all nodes;
  NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
}

for(WAIT_NODE_ID_ACK_STATE;
    BLSR_TBL_REQ)
{
  process blsr table request;
  send blsr table response;
  SAME_STATE;
}

for(WAIT_NODE_ID_ACK_STATE;
    NODE_ID_ACK)
{
  process node id ack;
  if all ack received {
    NEXT_STATE(IDLE_STATE);
  }
  SAME_STATE;
```

```
    }

    for(WAIT_NODE_ID_ACK_STATE;
        USER_ACCEPT_RM)
 5  {
      process ring map update;
      if accepted ring map is different from old one {
        nodeIdChangeQ = TRUE;
        send accepted ring map to all nodes;
10      NEXT_STATE(WAIT_RM_ACK_STATE);
      }
      SAME_STATE;
    }

15  for(WAIT_NODE_ID_ACK_STATE;
        RDY_TO_SWITCH_REQ)
    {
      // Ignore
      SAME_STATE;
20  }

    for(WAIT_NODE_ID_ACK_STATE;
        XC_ENABLE_SWITCH_REQ)
    {
25    xcEnableSwQ = TRUE;
      SAME_STATE;
    }

    for(WAIT_NODE_ID_ACK_STATE;
30      BLSR_TBL_RSP,  RM_ACK,
        RDY_TO_SWITCH_ACK, ENABLE_SWITCH)
    {
      STATE_ERROR;
    }
35
    ////////////////////////////////////////////////////
    // WAIT_ENABLE_SWITCH_RSP_STATE
    ////////////////////////////////////////////////////
    for(WAIT_ENABLE_SWITCH_RSP_STATE;
40      TOPOLOGY_CHANGE)
    {
      process OSPF change;
      if OSPF changes lead to new ring map {
        enableSwQ = TRUE;
45      NEXT_STATE(IDLE_STATE);
      }
      NEXT_STATE(SAME_STATE);
    }

50  for(WAIT_ENABLE_SWITCH_RSP_STATE;
        PROVISIONING)
    {
      if TCC is not active
      {
55      blsrActive = FALSE;
        readyToSwitch = FALSE;
        terminate all communication sessions
        NEXT_STATE(STANDBY_STATE);
      }
60
      SAME_STATE;
```

```
}

for(WAIT_ENABLE_SWITCH_RSP_STATE;
    LOCAL_XC_CHANGE)
{
  initialize local payload table;
  initialize squelch table;
  initialize add/drop tables;

  if number of nodes in ring > 1 {
    send cross-connect change to all nodes in ring;
    send blsr table request to all nodes in ring;
    NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
  }

  SAME_STATE;
}

for(WAIT_ENABLE_SWITCH_RSP_STATE;
    LOCAL_NODE_ID_CHANGE)
{
  process node id change;
  if number of nodes in ring > 1 {
    enableSwQ = TRUE;
    send node id to all nodes in ring;
    NEXT_STATE(WAIT_NODE_ID_ACK_STATE);
  }
  SAME_STATE;
}

for(WAIT_ENABLE_SWITCH_RSP_STATE;
    LOCAL_RING_ID_CHANGE)
{
  terminate all communication sessions;
  process latest OSPF changes;
  if  new ring Id lead to new ring map {
    NEXT_STATE(IDLE_STATE);
  }
  SAME_STATE;
}

for(WAIT_ENABLE_SWITCH_RSP_STATE;
    REMOTE_XC_CHANGE)
{
  terminate all communication sessions;
  send blsr table request to all nodes;
  NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
}

for(WAIT_ENABLE_SWITCH_RSP_STATE;
    REMOTE_NODE_ID_CHANGE)
{
  process node id change;
  send acknowledgement to node;
  SAME_STATE;
}

for(WAIT_ENABLE_SWITCH_RSP_STATE;
    REMOTE_RM_CHANGE)
{
  process ring map change;
```

```
      send acknowledgement to node;
      terminate all communication sessions;
      send blsr table request to all nodes;
      NEXT_STATE(WAIT_BLSR_TBL_RSP_STATE);
 5  }

    for(WAIT_ENABLE_SWITCH_RSP_STATE;
        RDY_TO_SWITCH_REQ)
    {
10    if(readyToSwitch) {
        send acknowledgement to node;
      }
      SAME_STATE;
    }
15
    for(WAIT_ENABLE_SWITCH_RSP_STATE;
        BLSR_TBL_REQ)
    {
      process blsr table request;
20    send blsr tables to node;
      SAME_STATE;
    }

    for(WAIT_ENABLE_SWITCH_RSP_STATE;
25      RDY_TO_SWITCH_ACK)
    {
      process ready to switch ack;
      if(received ready to switch ack from all nodes) {
        if(!enableSwitchSentToXc && xcReadyToSwitch) {
30        send ready to switch message to XCON;
        }
        NEXT_STATE(IDLE_STATE);
      }
      else {
35      SAME_STATE;
      }
    }

    for(WAIT_ENABLE_SWITCH_RSP_STATE;
40      XC_ENABLE_SWITCH_REQ)
    {
      xcEnableSwQ = TRUE;
      SAME_STATE;
    }
45
    for(WAIT_ENABLE_SWITCH_RSP_STATE;
        USER_ACCEPT_RM)
    {
      process ring map update;
50    if accepted ring map is different from old ring map {
        send ring map to all nodes in ring;
        NEXT_STATE(WAIT_RM_ACK_STATE);
      }
      SAME_STATE;
55  }

    for(WAIT_ENABLE_SWITCH_RSP_STATE;
        BLSR_TBL_RSP, NODE_ID_ACK,
        RM_ACK, ENABLE_SWITCH)
60  {
      STATE_ERROR;
```

}